Scott Shrout

Programming Assignment 5

Due 11/21/2017

## Abstract

In this Programming Assignment, we are tasked with making modifications to previously submitted Programming Assignment 4 to use a binary search tree (BST) to store and search our dictionary as opposed to a linked list. Beyond this, the assignment is largely unchanged: we are to read in the dictionary.txt file and add each word found to the binary search tree associated with the first letter of the word. After each word is read into the dictionary, we are ready to begin traversing our text file oliver.txt. While reading the file (and making a somewhat concerted effort to strip out unnecessary characters), each word is searched for in the corresponding binary search tree. Counters are in place and updated appropriately to track the occurrence of words found / not found as well as the number of comparisons needed for each type. Unlike the prior assignment, we are asked to output the words that were not found.

The use of a binary search tree in place of a linked list was expected to demonstrate a reduction in comparison operations (and associated time) though I found the difference to be somewhat more dramatic than I anticipated. While the spell check function in Programming Assignment 4 required on average 3,554 comparisons for words found and 7,438 for words not found, the binary search tree required only an average of 16 comparisons per word found and 14 for words not found. Though I made a few tweaks in this assignment to strip out additional unnecessary characters (increase word recognition), the number of words / not found did not differ significantly.

There are several advantages of using a binary search tree in this type of application that were apparent before beginning the assignment. Firstly, the tree is in a logical (alphabetic) order while the linked list was only in an order determined by the order in which words were added to the dictionary. Secondly, the search operation begins by traversing only half of the list based upon whether the value being searched for is greater than or equal to the root element. In addition, the search continues by evaluating each element and moving to an element to the left or right of the current element based upon its comparison to the searched term. It is therefore able to terminate the search (with a false result) if the element moved to is null. These differences pose a significant advantage for the binary search tree when it comes to searching as it, at worst, only has to traverse approximately half of the list and is able to detect and stop when it is clear the element is not in the list. The linked list, on the other hand, at worst requires a complete traversal of the list and because elements are not sorted, cannot terminate the search prior to reaching the desired element or the end of the list.

While the binary search tree sees significant advantage in searching operations, it does require more significant effort to insert new elements as opposed to the linked list. While the linked list inserts new elements at the end of the list, the binary search tree attaches each new element to the left or right of an existing element (establishing its order). This requires a traversal similar to that of its search operation, comparing the value to be inserted to the root and then each subsequently traversed element.

Overall, I found this assignment interesting as it clearly demonstrated that there is significant variance in terms of efficiency among generic data structures that make them much more ideal for some applications than others. In regards to the linked list, it presents some advantages in terms of efficiency when adding new elements to the list as well as when searching for elements that happen to lie at the beginning of the list. The binary search tree, on the other hand, requires some additional effort when inserting elements, but presents significant advantage

when searching for elements especially when they lie near the center of the list (though it likely enjoys advantage elsewhere as well, as it is sorted).